# contexto problemático:

Se sabe que las naves de Marte están ubicadas estratégicamente de forma secreta e invisibles a las tropas de Venus. Cada nave posee unas coordenadas *x* y *y*, como parte de su posición en el tablero de batalla. La única forma de descifrar su ubicación y destruirlas es utilizando la técnica de multiplicación de matrices para encontrar las coordenadas exactas.

La matriz de batalla, donde se ven reflejadas las posiciones de las naves de la flota marciana, se encuentra compuesta por números enteros positivos. Si el número presente dentro de alguna celda del tablero es un número primo, en dicha posición se encuentra presente una nave. Como la matriz que poseen los altos mandos venusianos tan sólo representa la ubicación de la flota de Marte en guerras pasadas, se ha de multiplicar por una matriz de coeficientes que ha podido descubrir el servicio secreto de Venus. Dicha multiplicación dará lugar a una nueva matriz de enteros que mostrará la ubicación de las naves en la actualidad, al igual que su cantidad real.

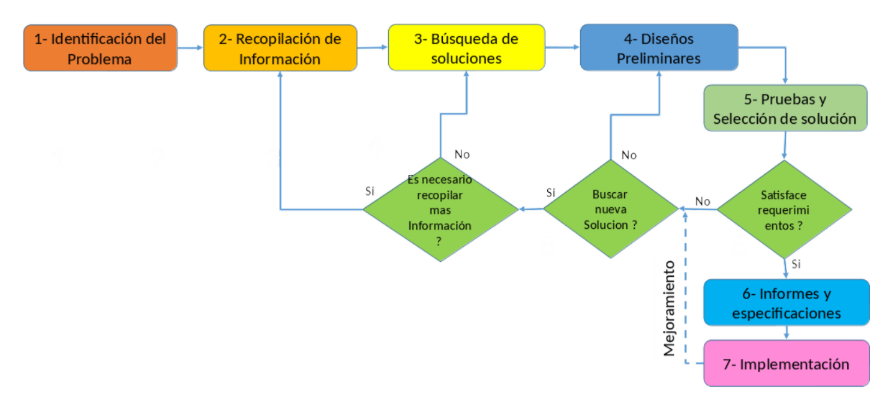
# Desarrollo de la solución:

Para resolver la situación anterior se eligió el Método de la Ingeniería para desarrollar la solución siguiendo un

enfoque sistemático y acorde con la situación problemática planteada.

Con base en la descripción del Método de la Ingeniería del libro “Introduction to Engineering” de Paul Wright,

se definió el siguiente diagrama de flujo, cuyos pasos seguiremos en el desarrollo de la solución.



# 

# paso 1: identificación del problema

Identificacion de sintomas y necesidades

* las tropas venusianas requieren saber la ubicación de las naves marcianas en la matriz de batalla
* el programa deber ser capaz de identificar la posición actual de las naves marcianas en la matriz de batalla
* se requiere que se pueda ingresar la matriz de posiciones anteriores y/o la de coeficientes de movimientos
* el programa debe ser capaz de multiplicar 2 o más matrices de manera rápida
* el programa de identificar las naves enemigas (las cuales son números primos) y representarlas en pantalla

Definición del problema

los venusianos requieren un programa que les permita el ingreso de matrices (de batalla y/o de coeficientes) y que les retorne una nueva matriz con las posiciones actuales de la naves enemigas identificando y representando todos los números primos como naves enemigas

|  |  |
| --- | --- |
| nombre | R1. Multiplicar Matrices |
| resumen | el programa deberá ser capaz de multiplicar un número n de matrices |
| entrada | matrices a multiplicar |
| salida | matriz resultante |

|  |  |
| --- | --- |
| nombre | R2. Mostrar naves |
| resumen | el programa deberá ser capaz de representar los número primos dentro de una matriz de enteros como naves |
| entrada | matriz |
| salida | matriz con naves |

# paso 2: recopilación de información

Matriz

Una matriz es un arreglo [bidimensiona](https://es.wikipedia.org/wiki/Bidimensional)l de [números](https://es.wikipedia.org/wiki/N%C3%BAmero). Dado que puede definirse tanto la suma como el producto de matrices, en mayor generalidad se dice que son elementos de un [anillo](https://es.wikipedia.org/wiki/Anillo_(matem%C3%A1ticas)). Una matriz se representa por medio de una letra mayúscula (A,B, …) y sus elementos con la misma letra en minúscula (a,b, …), con un doble subíndice donde el primero indica la fila y el segundo la columna a la que pertenece.

[https://es.wikipedia.org/wiki/Matriz\_(matemáticas)](https://es.wikipedia.org/wiki/Matriz_(matem%C3%A1ticas))

En [programación](https://es.wikipedia.org/wiki/Programaci%C3%B3n), se le denomina vector, formación o arreglo (en inglés array)[1](https://es.wikipedia.org/wiki/Vector_(inform%C3%A1tica)#cite_note-1)​ a una zona de almacenamiento contiguo que contiene una serie de elementos del mismo tipo, los elementos de la matriz.[2](https://es.wikipedia.org/wiki/Vector_(inform%C3%A1tica)#cite_note-2)​ Desde el punto de vista lógico una matriz se puede ver como un conjunto de elementos ordenados en fila (o filas y columnas si tuviera dos dimensiones).

<https://es.wikipedia.org/wiki/Vector_(inform%C3%A1tica)>

Una matriz es una tabla cuadrada o rectangular de datos (llamados elementos) ordenados en filas y columnas, donde una fila es cada una de las líneas horizontales de la matriz y una columna es cada una de las líneas verticales. A una matriz con m filas y n columnas se le denomina matriz m-por-n (escrito m×n). Las dimensiones de una matriz siempre se dan con el número de filas primero y el número de columnas después.

<https://sites.google.com/site/algebralinealmoralescamacho/u2-matrices/2-1-definicion-de-matriz-notacion-y-orden>

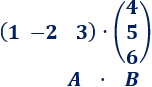
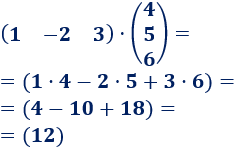
Multiplicación de matrices

Solo se pueden multiplicar 2 matrices si sus dimensiones son compatibles, lo que significa que el número de columnas de la primera matriz es igual al número de filas de la segunda matriz. Si A es una matriz a x b , y B es una matriz b x c, el producto de AB es una matriz a x c.

<https://www.varsitytutors.com/hotmath/hotmath_help/spanish/topics/matrix-multiplication>

El resultado de la multiplicación A⋅B es la suma del primer elemento de A por el primero de

B más el segundo elemento de A por el segundo de B más el tercer elemento de A por el tercero de B:

* El producto de matrices no es conmutativo. Es decir, el producto de matrices AB suele ser distinto a BA

<https://www.problemasyecuaciones.com/matrices/multiplicar-matrices-producto-matricial-ejemplos-explicados-propiedades-matriz.html>

Usted solo puede multiplicar dos [matrices](https://www.varsitytutors.com/hotmath/hotmath_help/spanish/topics/matrices.html) si sus [dimensiones](https://www.varsitytutors.com/hotmath/hotmath_help/spanish/topics/matrix-dimensions.html) son [compatibles](https://www.varsitytutors.com/hotmath/hotmath_help/spanish/topics/compatible-matrices.html) , lo que significa que el número de columnas en la primera matriz es igual al número de renglones en la segunda matriz. Si A es una matriz a × b y B es una matriz b × c , el producto AB es una matriz a × c .

La definición de la multiplicación de matrices indica una multiplicación renglón-por-columna, donde las entradas en el renglón i th de A son multiplicadas por las entradas correspondientes en el renglón j th de B y luego se suman los resultados.

La multiplicación de matrices NO es conmutativa. Si ni A ni B son una matriz identidad, AB ≠ BA

<https://www.varsitytutors.com/hotmath/hotmath_help/spanish/topics/matrix-multiplication>

Identificacion de numeros primos

* Los números primos son aquellos números los cuales son solo divisible por 1 y por el mismo

<https://okdiario.com/curiosidades/numeros-primos-1125886>

###### ¿Qué son los números primos?

Los números primos son aquellos que solo son divisibles entre ellos mismos y el 1.

<https://www.smartick.es/blog/matematicas/numeros/numeros-primos-y-numeros-compuestos/>

Las Matemáticas diferencian entre dos tipos de números: los primos y los compuestos. En este sentido los números primos serían aquellos que sólo presentan dos divisores, es decir, que sólo pueden dividirse entre el número 1 y entre ellos mismos. Mientras que los números compuestos son aquellos que pueden dividirse por más de dos divisores.

[**https://educacion.elpensante.com/ejemplo-de-como-identificar-numeros-primos-y-compuestos/**](https://educacion.elpensante.com/ejemplo-de-como-identificar-numeros-primos-y-compuestos/)

# paso 3: búsqueda de soluciones creativas

Alternativa 1:

En primera instancia se deberá ingresar las matrices a multiplicar y comprobar que se pueden multiplicar mediante un algoritmo que verifique la condición de multiplicación de matrices, despues de eso identificar los números primos en la primera matriz para ver cuál fue la última posición de los marcianos y en base a ella, hallar la nueva posición de los marcianos y notificar su posición mediante una alerta la cual dará su i-ésima posición y j-ésima posición en la matriz resultante.

Alternativa 2:

Pedir en primera instancia el número de matrices a multiplicar y después el tamaño de cada una de las matrices y mediante un algoritmo comprobar su compatibilidad para ir ingresando uno a uno los números deseados, cuando estas se multipliquen mostrar la i-ésima posición y j-ésima posición de las naves marcianas en la matriz de batalla actual mediante un algoritmo que identifique si es numero primo. Cuando el usuario decida introducir nuevas matrices preguntar si desea que la matriz de la batalla recién encontrada se agregada a las matrices a multiplicar, si es así se anexara a las matrices por multiplicar, y en caso de que no sea así se agregaran las nuevas matrices a multiplicar.

Alternativa 3:

Ingresar el tamaño de cada una de las matrices a multiplicar, para luego agregar uno a uno todos los valores dentro de dichas matrices y realizar el proceso respectivo de multiplicación e identificación de números primos

Alternativa 4:

EL programa no pedira ningun tipo de dato de entrada a parte de la información sobre si habrá o no valores repetidos dentro de las matrices y con esta información auto generará las matrices de batalla anterior y la de coeficientes asegurándose de que ambas puedan multiplicarse entre sí y de que la matriz de batalla anterior tenga por lo menos una nave (número primo)

Alternativa 5:

Utilizar como algoritmo de multiplicación de matrices el algoritmo de Strassen para casos en los que las matrices que entren al programa sean muy grandes. Se permitirá tener más de 2 matrices y comprobar su compatibilidad además de tener la función de rellenado aleatorio o ingresando los dígitos uno a uno para posteriormente evaluar cuales son las naves marcianas buscando los números primos en la matriz de batalla resultante que también podrá ser utilizada posteriormente en futuros cálculos.

encontrado en: <https://en.wikipedia.org/wiki/Strassen_algorithm#Asymptotic_complexity>

Alternativa 6:

En primera instancia ingresar el tamaño de las matrices y verificar que se puedan multiplicar, luego ingresar cada uno de los datos que van dentro de las matrices tanto la de batalla anterior como la coeficientes y que un solo método se encargue tanto de la multiplicación de las matrices como de la verificación de los números primos.

Alternativa 7:

Usar el algoritmo de divide y vencerás como algoritmo principal para la multiplicación de matrices debido a que la división de matrices en estructuras más simples y recursivas hace más eficiente su recorrido y multiplicación

encontrado en: https://webdiis.unizar.es/asignaturas/EDA/ea/slides/3-Divide%20y%20venceras.pdf

# paso 4: transición de las ideas a los diseños preliminares

Alternativa 1:

* El método no permite ingresar más de 2 matrices para ser multiplicadas.
* primero se deben deben llenar antes la matrices antes de verificarlas y por lo tanto tendrá un tamaño limitado en matriz.
* Esta alternativa no guarda la información de la última batalla.
* tiene un número limitado de tamaño y por ello se puede resolver más el problema ya que lo valores pueden son más limitados.
* se mostrará la i-ésima y j-ésima posición de las naves marcianas en la matriz de batalla encontrada.
* Esta alternativa queda descartada debido a que no cumple con requisitos fundamentales para los clientes venusianos.

Alternativa 2:

* Permite ingresar el tamaño de la matriz primero y por eso se puede comprobar más rápido, pero da demasiada libertad a la hora de escoger el tamaño y por ello se pueden tener entradas demasiado grandes para el equipo.
* Como entra primero el número de matrices con sus respectivos tamaños permite que la comprobación de su multiplicación se más instantánea.
* se mostrará la i-ésima y j-ésima posición de las naves marcianas en la matriz de batalla encontrada.
* permite utilizar la matriz de batalla encontrada recientemente en futuras entradas.

Alternativa 3:

* Esta alternativa no implica un método innovador a la hora de multiplicar las matrices, simplemente las multiplicará en el orden sucesivo de n x n.

Alternativa 4:

* Esta alternativa tendrá un número fijo en el tamaño de matrices y en su cantidad, y por ello el programa será bastante eficiente pero a su vez muy limitado.
* En cuestión de dificultad algorítmica se implementa de manera muy sencilla.
* No hay libertad en escoger qué dígitos deberían tener las matrices.

Alternativa 5:

* El algoritmo de Strassen es un algoritmo comprobado que será muy práctico en el desarrollo del programa.
* Esta alternativa ya ha sido comprobado de ser un algoritmo muy eficiente y por ello se pueden multiplicar más de 2 matrices con facilidad.

Alternativa 6:

* El método sería fácil de implementar, pero tendría una baja cohesión y por ello la reutilización del código para crear una nueva versión del programa sería más complicada y en un caso más extremo sería imposible.
* Su implementación algorítmica y cálculo de complejidad temporal sería más sencilla al tener todo en un solo lugar.
* No sería eficiente al tener un método demasiado extenso.

Alternativa 7:

* el programa resultará considerablemente más rápido y eficiente a la hora de multiplicar las matrices
* las estructuras recursivas en caso de una matriz demasiado grande pueden significar un problema dependiendo del dispositivo
* en términos temporales es más difícil de calcular el tiempo se repite una estructura recursiva

# paso 5: evaluación y selección de la mejor solución

Criterios

* Criterio A: Precisión de la solución.
  + [2] Exacta
  + [1] Aproximada
* Criterio B:Eficiencia en el algoritmo de multiplicación y identificación de números primos.
  + [4] Constante
  + [3] Mayor a constante
  + [2] Logaritmica
  + [1] Lineal
* Criterio C: Facilidad en la implementación algorítmica.
  + [2] Compatibles con las operaciones aritméticas básicas de un equipo de computo
  + [1] No compatible completamente con las operaciones aritméticas básicas de un equipo de computo moderno.
* Criterio D: Completitud de la solución
  + [3] Todas y más
  + [2] Más de una, pero no todas
  + [1] sólo una o ninguna
* Criterio E: Implementación dado el tiempo.
  + [3] Sobrante
  + [2] Óptimo
  + [1] Escaso

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Criterio | Criterio A | Criterio B | Criterio C | Criterio D | Criterio E | Total |
| Alternativa 2 | Exacta (2) | Lineal (1) | Compatible (2) | Más de una pero no todas (2) | Sobrante (3) | 10 |
| Alternativa 3 | Exacta (2) | Lineal (1) | Compatible (2) | Más de una pero no todas(2) | Sobrante (3) | 10 |
| Alternativa 4 | Aproximada (1) | Lineal (1) | Compatible  (2) | Solo una (1) | Sobrante(3) | 8 |
| Alternativa 5 | Exacta (2) | Logarítmica (2) | Compatible (2) | Todas y más (3) | Óptimo (2) | 11 |

Resultados: La alternativa 5 es la más viable en vista de que cumple con todos las necesidades del cliente, tiene un algoritmo bastante eficiente en comparación con los demás, y el tiempo de implementación es bastante óptimo.

# paso 6: preparación de informes y especificaciones

Especificación del problema

1.

Problema: Multiplicación de matrices

Entradas: Dos matrices de tamaño nxm

Salida: Matriz resultante

2.

Problema: identificacion de numero primos en matriz

Entrada: matriz de enteros

Salida: Matriz de batalla

Consideraciones

1. las matrices deben poder multiplicarse entre sí.
2. el objeto matriz de batalla es un objeto implementado por nosotros
3. la identificación de números primos hace referencia a convertir la posición en la que se encuentra un número primo en una imagen

Pseudocodigo del algoritmo

1. multiply(A[][],B[][])

R[a][b]

for(I = 0 to N){

for(J = 0 to N){

R[I][J]=0

for(K = 0 to N){

R[I][J] += A[I][K]\*B[K][J]

}

}

}

1. itSPrime(A[][])

Battle = [n][m]

for( I = 0 to A.length){

for( J = 0 to A[I].length){

n = 0

sum = 0

for(v=0;v<A[i][j];v++){

if(A[i][j] mod v=0){

sum = sum +1

}

}

if(sum=1){

Battle[i][j] = ufo

}

}

}

# paso 7: implementación del diseño

|  |  |  |
| --- | --- | --- |
| Nombre Escenario | Clase | Escenario |
| setupScenary1 | SpaceWarControllerTest |  |
| setupScenary2 | SpaceWarControllerTest |  |

Casos de prueba

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Objetivo de la prueba: Comprobar que 2 matrices se multiplican | | | | |
| Clase | Método | Escenario | Valores de entrada | Resultado |
| SpaceWarControllerTest | simplyMultiplyTest | setupScenary1 | ninguno | Las matrices se multiplican satisfactoriamente. |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Objetivo de la prueba: Comprobar que el programa identifica naves enemigas | | | | |
| Clase | Método | Escenario | Valores de entrada | Resultado |
| SpaceWarControllerTest | primeNumberTest | setupScenary2 | ninguno | Los números primos son identificados correctamente en la matriz. |

**Analisis de complejidad Temporal**

1. randomRepitedNumbers()

|  |  |  |  |
| --- | --- | --- | --- |
| # | Linea | Constantes | Valor |
| 1 | Random rnd=new Random(); | C1 | 1 |
| 2 | for (int i = 0; i < matrix.length; i++) { | C2 | n+1 |
| 3 | for (int j = 0; j < matrix[i].length; j++) { | C3 | +1 |
| 4 | int rand = rnd.nextInt(100)+1; | C4 |  |
| 5 | matrix[i][j]=rand; | C5 |  |

At(n)= = O()

1. randomUnRepitedNumbers(int columns, int rows)

|  |  |  |  |
| --- | --- | --- | --- |
| # | Linea | Constantes | Valor |
| 1 | int[] randoms = new int[columns\*rows]; | C1 | 1 |
| 2 | randoms[0] = (int)(Math.random()\*100); | C2 | 1 |
| 3 | for (int i = 1; i < randoms.length; i++) | C3 | n+1 |
| 4 | randoms[i]=(int)(Math.random()\*100) | C4 | n |
| 5 | for (int j = 0; j < 1; j++) | C5 |  |
| 6 | if(randoms[i]==randoms[j]) | C6 | +n |
| 7 | i-- | C7 | +n-1 |
| 8 | for (int i = 0; i < matrix.length; i++) | C8 | n+1 |
| 9 | for (int j = 0; j < matrix.length; j++) | C9 | +n+1 |
| 10 | for (int k = 0; k < randoms.length; k++) | C10 | ++n |
| 11 | matrix[i][j] = randoms[k]; | C11 | ++n-1 |

At(n)= = O(

1. simpleMultiply(boolean repeat2, int rows, int columns, int coeficientRow, int coeficientColumn)

{

;

{

{

for (int k = 0; k < battleMatrix[0].length; k++) {

battleMatrix[i][j] += oldbattle[i][k]\*coefients[k][j];

}System.out.println("se agrego a la matriz un dato");

}

}

}else {

System.out.println("esa monda no se puede");

}

|  |  |  |  |
| --- | --- | --- | --- |
| # | Linea | Constantes | Valor |
| 1 | BattleMatrix battle = new BattleMatrix(repeat2, rows, columns) | C1 | 1 |
| 2 | int[][] coefients = coeficientMatrix(coeficientRow, coeficientColumn) | C2 | 1 |
| 3 | int[][] oldbattle = battle.getMatrix(); | C3 | 1 |
| 4 | int[][] battleMatrix = new int[rows][coeficientColumn] | C4 | 1 |
| 5 | if(operable(rows, coeficientColumn)==true) | C5 | 1 |
| 6 | for (int i = 0; i < battleMatrix.length; i++) | C6 | n+1 |
| 7 | for (int j = 0; j < battleMatrix[i].length; j++) | C7 |  |
| 8 | for (int k = 0; k < battleMatrix[0].length; k++) | C8 |  |
| 9 | battleMatrix[i][j] += oldbattle[i][k] \* coefients[k][j]; | C9 |  |

O